山东大学
SHANDONG UNIVERSITY

崇新学堂

2024 — 2025 学年第一学期

实验报告

课程名称：Introduction to EECS Lab

实验名称：Homework 2: Heads Up!

学生姓名：胡君安、陈焕斌、黄颢

实验时间：2024 年 10 月 31 日

Homework 2: Heads Up!

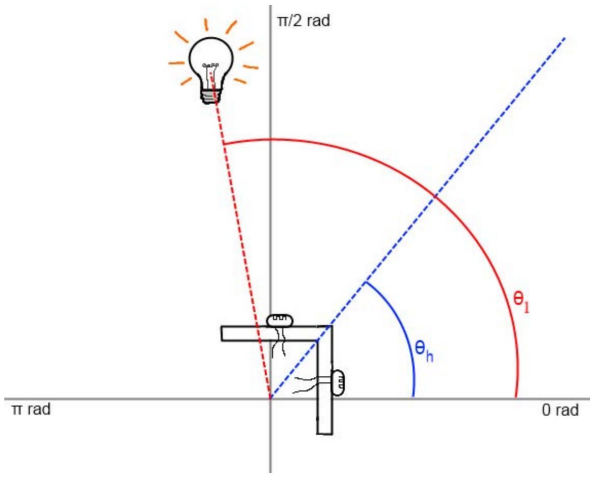
- Lab Report

Introduction

In this experiment, we apply techniques developed in Design Labs 4 and 5 to construct and analyze a model for a system that will be revisited in Design Labs 7 through 9. This system controls a motor to rotate a robot “head” toward a light source. The setup comprises three primary components:

- **A light sensor**, which detects the position of the light relative to the head
- **A motor**, which turns the head in the direction of the light
- **A control circuit**, which regulates the motor (and will be constructed in later labs)

We can describe the system as follows: the light source has an absolute angular position, denoted by θ_l, and the head also has its own absolute angular position, θ_h. The objective is to design a system that adjusts θ_h to match θ_l, ensuring the head consistently aligns with the light. The relationship between θ_h and θ_l is illustrated below.



Light Tracker

Symbolic description

- Θ_l : the absolute angular position of the light, a signal whose samples are $\theta_l[n]$.
- Θ_h : the absolute angular position of the head, a signal whose samples are $\theta_h[n]$.
- E : the error signal $(\Theta_l - \Theta_h)$.
- V_s : the voltage produced by the light direction sensor, a signal whose samples are $v_s[n]$.
- I_m : the current flowing through the motor, a signal whose samples are $i_m[n]$.
- Ω_h : the motor's angular *velocity*, a signal whose samples are $\omega_h[n]$.
- A_h: the angular acceleration of the motor, a signal whose samples are $\alpha_h[n]$.
- V_c : motor control voltage, a signal whose samples are $v_c[n]$.
- V_b : the back-EMF of the motor, a signal whose samples are $v_b[n]$.

Building the model

According to the experimental requirements, we have the following relationship formula between physical quantities:

$$
\begin{aligned}
V_b[n] &= k_b w_n[n] \\
i_m[n] &= \frac{V_c[n] - V_b[n]}{r_m} \\
V_s[n] &= k_s\theta_i[n] - k_s\theta_n[n] \\
\alpha_n[n] &= k_m i_m[n] \\
\alpha_n[n]T &= w_n[n] - w_n[n-1] \\
V_c[n] &= k_c V_s[n]
\end{aligned}
$$

Modeling the Sensor and Controller

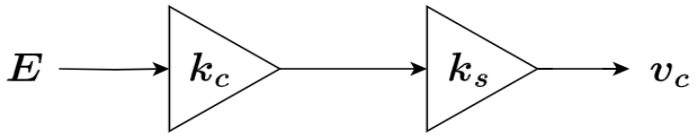
In this section, we aim to model the sensor and controller components of our system, which is designed to control a motor to turn a robot head towards a light source. The system uses a simple proportional controller to generate the control voltage (Vc) based on the sensor signal (Vs). The sensor signal (Vs) is proportional to the error (E), which is the difference between the light's angular positionnand the head's angular position.

System function:

$$
\begin{cases}
V_s[n] = k_s(\theta[n] - \theta_h[n]) = k_s\theta[n] \\
V_c[n] = k_c V_s[n]
\end{cases}
$$

$$
H_{cs} = \frac{V_c}{E} = k_c k_s
$$

System diagram:



Write the corresponding code according to the system diagram.

```
Python
def controllerAndSensorModel(k_c):
   return sf.Gain(k_c*k_s)
```

Modeling the Plant

Our plant takes as input Vc (the output of the controller), and generates output Ih.

Integrator

An integrator takes as its input the motor's angular velocity Ωh, and outputs the motor's angular position Θh.

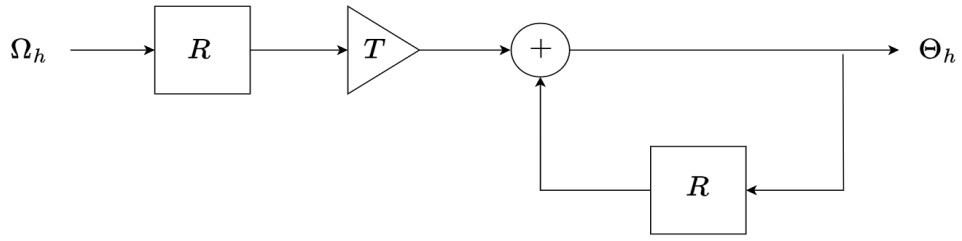
System function:

$$\omega_h[n-1] \cdot T = \theta_h[n] - \theta_h[n-1]$$

$$R\Omega_h \cdot T = \Theta_h - R\Theta_h$$

$$H_1 = \frac{RT}{1-R}$$

System diagram:



System diagram of Integrator

Write the corresponding code according to the system diagram.

```
Python
def integrator(T):
   return sf.Cascade(sf.Cascade(sf.Gain(T),sf.R()),
            sf.FeedbackAdd(sf.Gain(1),sf.R()))
```

Motor

A motor takes Vc as input and generates Ω as output.

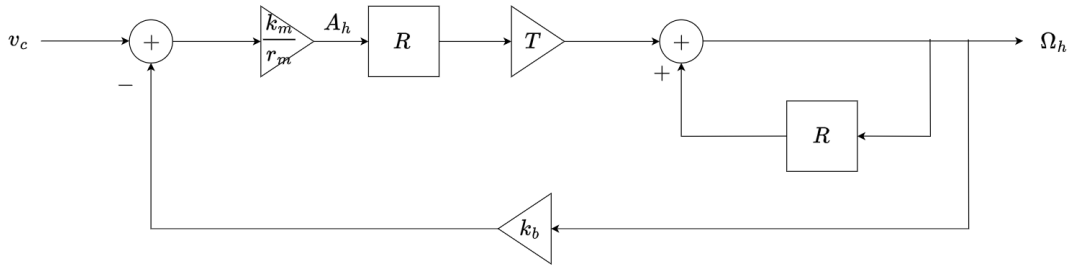
System function:

$$\begin{cases} \frac{V_c[n]-k_b\omega_h[n]}{r_m} = \frac{\alpha_h[n]}{Tk_m} \\ \alpha_h[n]\cdot T = \omega_h[n]-\omega_h[n-1] \end{cases}$$

$$\Rightarrow \frac{k_m}{r_m}(V_c - k_b\omega_h[n]) = \alpha_h[n]$$

$$H_2 = \frac{k_m RT}{r_m(1-R)+k_m k_b RT}$$

System diagram:



System diagram of Motor

Write the corresponding code according to the system diagram.

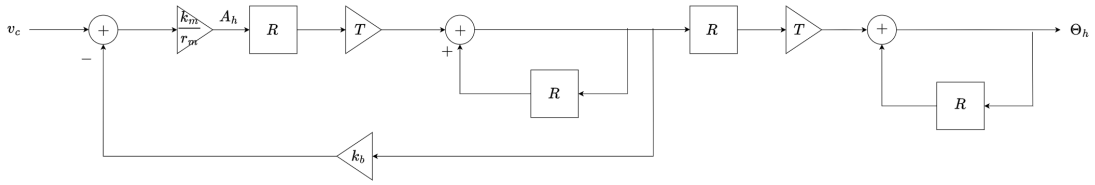
```
Python
def motorModel(T):
   return sf.FeedbackSubtract(
                              sf.Cascade(
                                         sf.Cascade(sf.Gain(k_m/r_m),
sf.R()),
                                         sf.Cascade(sf.Gain(T),
sf.FeedbackAdd(sf.Gain(1), sf.R()))
                                                   ),
                              sf.Gain(k_b)
                              )
```

The Combined Plant

System function:

$$H_{\text{plant}} = H_1 \cdot H_2 = \frac{k_m R^2 T^2}{r_m(1-R)^2 + k_m k_b T R(1-R)}$$

System diagram:



System diagram of Plant

Write the corresponding code according to the system diagram.

```
Python
def plantModel(T):
   return sf.Cascade(motorModel(T),integrator(T))
```

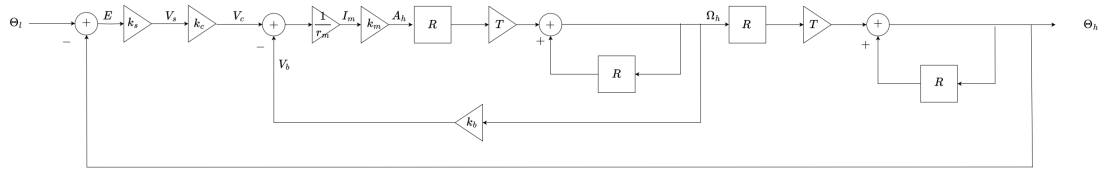
The Whole System

System function:

$$H = \frac{H_{cs} \cdot H_{\text{plant}}}{1 + H_{cs} \cdot H_{\text{plant}}}$$

$$H = \frac{k_m k_c k_s R^2 T^2}{r_m(1-R)^2 + k_m k_b T R(1-R) + k_m k_c k_s R^2 T^2}$$

System diagram:



System diagram of the whole system

```
Python
def lightTrackerModel(T,k_c):
   return sf.FeedbackSubtract(sf.Cascade(controllerAndSensorModel(k_c),
                                         plantModel(T)),sf.Gain(1))
```

Analyzing the System

Based on the overall system function, we can calculate the poles of the system.

$$P_0 = \frac{-k_m k_b T + 2r_m \pm T\sqrt{k_m^2 k_b^2 - 4r_m k_m k_c k_s}}{2r_m}$$

Substitute the corresponding data:

- $k_m = 1000 \frac{\frac{rad}{sec^2}}{Amp}$
- $k_b = 0.5 \frac{volts}{\frac{rad}{sec}}$
- $k_s = 5 \frac{volts}{rad}$
- $r_m = 20$ ohms

We can find the expression of pole p with respect to k_c:

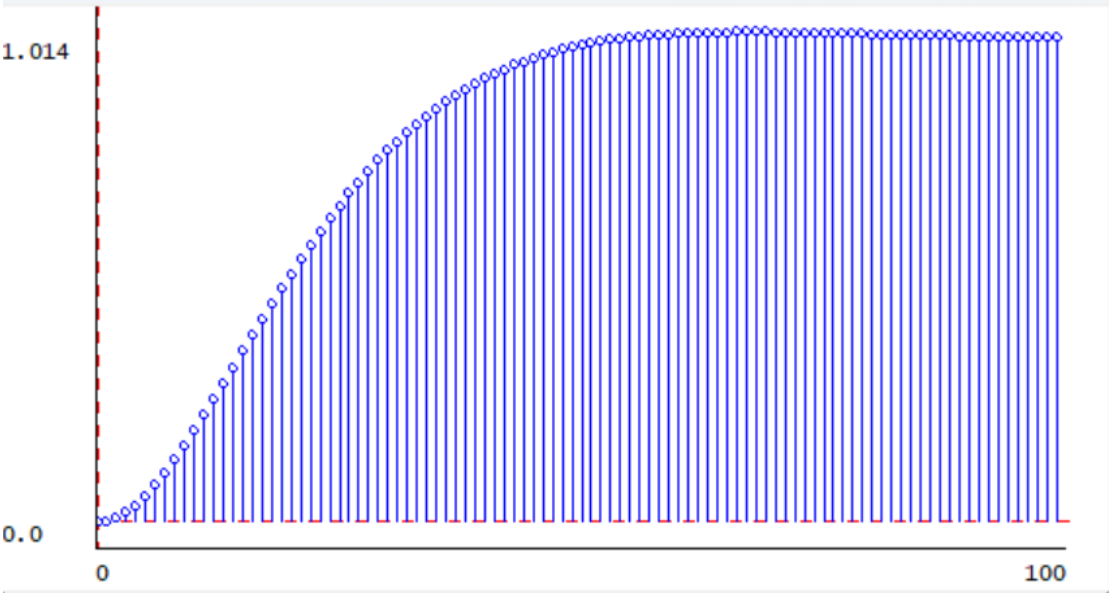
$$p_0 = 0.9375 \pm 0.0125\sqrt{25 - 40k_c}$$

By using the optimize module, we can find the optimal k_c value: such that the dominant pole is minimized, and the system reaches the steady state fastest and stably.

```
Python
def k_cFinder(T, k_cmin, k_cmax, numsteps):
    print optimize.optOverLine(lambda k_c: abs(lightTrackerModel(T,
k_c).dominantPole()), k_cmin, k_cmax, numsteps)
k_cFinder(0.005, -10, 10, 200)
```

Running the program, k_c = 93774996667555244.

generating a plot of the behavior resulting from the system starting at rest with a unit step signal as input.



k=0.93775

When T = 0.005s, investigate the impact of different values of kc on the system.

Depending on the location and type of poles, the response of the system can manifest as **monotonic convergence** or **oscillatory convergence** .

Monotonically converges when, in a discrete system, the pole p satisfies | p | < 1 and the pole is a real number.

According to the extreme expression of the system:

$$p = 0.9375 \pm 0.0125\sqrt{25 - 40k_c}$$

Monotonic convergence is required, and the pole must be real and satisfy | p | < 1. This requires the pole to be real **,** that is, the expression inside the square root is non-negative, that is:

$$\sqrt{25 - 40k_c} \geq 0 \Rightarrow k_c \leq 0.625$$

If all the poles p satisfy | p | < 1 and the poles are located within the unit circle, then the conditions for monotonic convergence are:

$$0 < k_c \leq 0.625$$

The condition for oscillation convergence is that in a discrete system, the pole p satisfies | p | < 1 and the pole is a complex number.

In the given system, the pole expression of the system is:

$$p = 0.9375 \pm j \times 0.0125\sqrt{40k_c - 25}$$

Therefore, the conditions for oscillation convergence are:

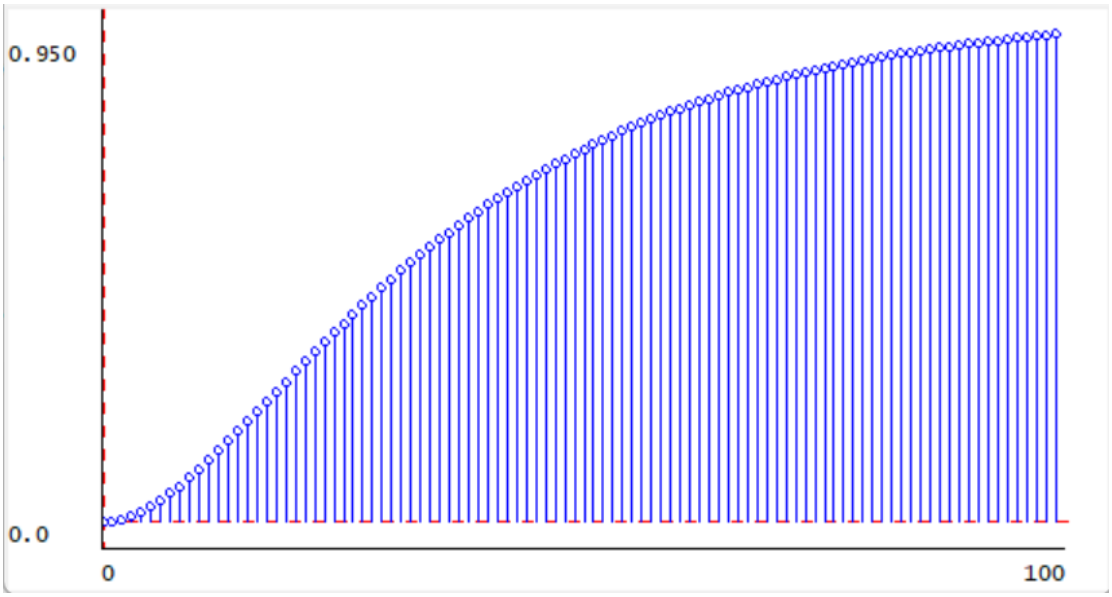
$$0.625 < k_c < 20$$

Within this range, the poles of the system are complex conjugate pairs with a modulus length less than 1, ensuring that the system response converges to steady state in an oscillating manner.

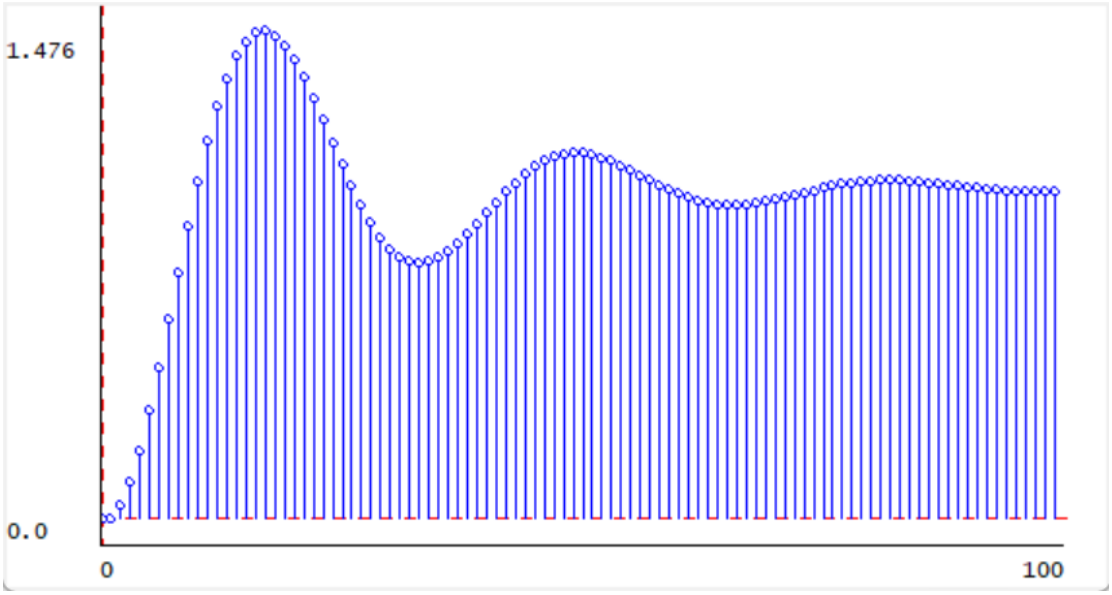
The condition for system instability is that the pole is outside the unit circle or the modulus length is equal to or greater than 1. Therefore, the condition for system instability is:

$$k_c \geq 20 \quad \text{or} \quad k_c \leq 0$$

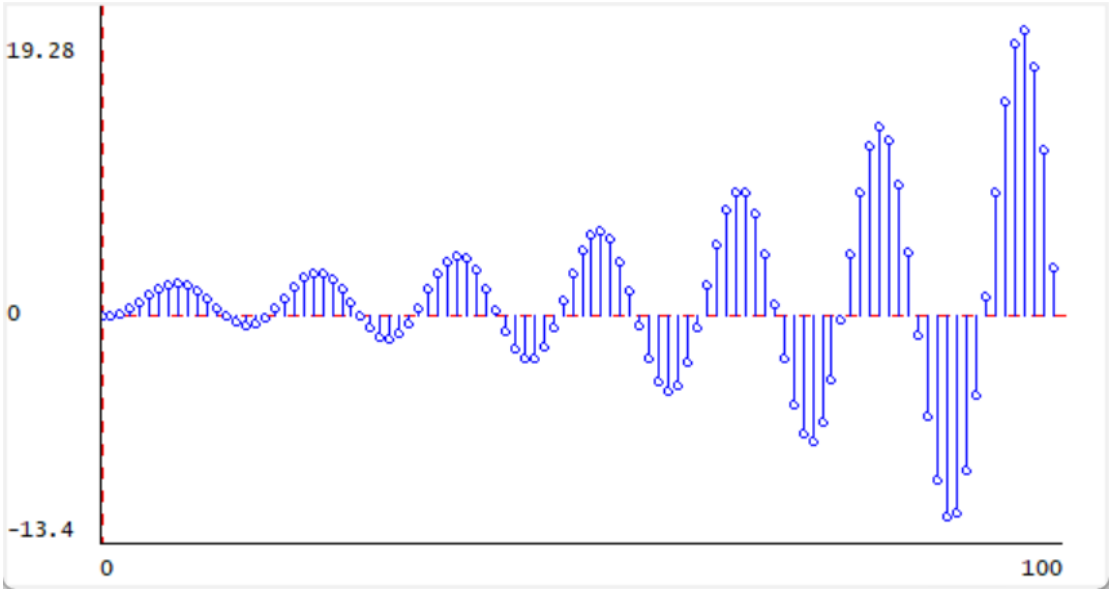
Below, we will take k = 0.5, k = 6, and k = 30 for each range to draw the corresponding system response graph. The results are as expected.



k=0.5

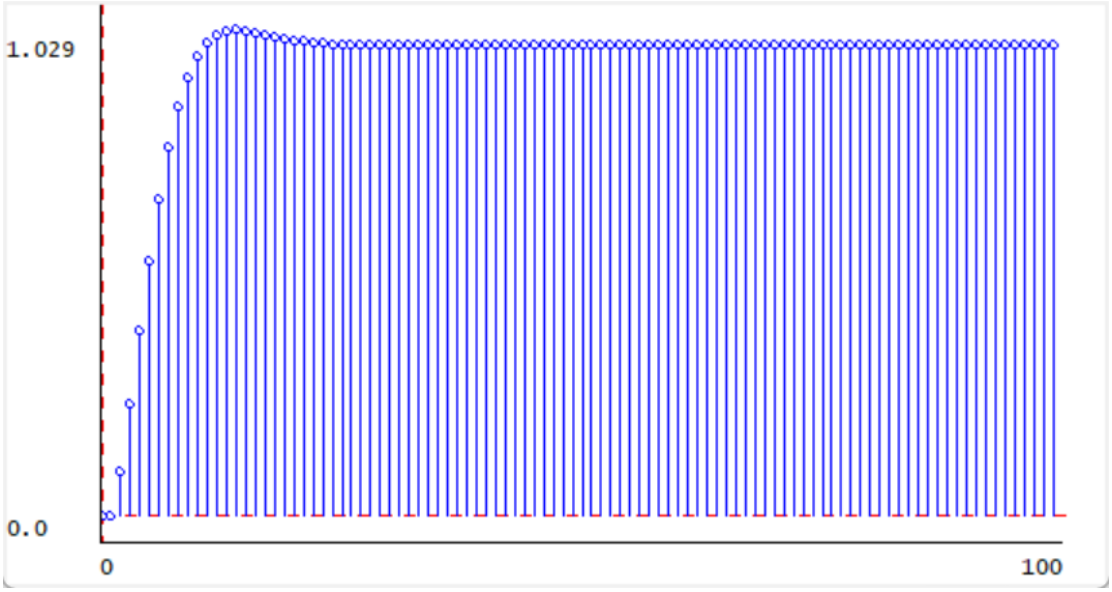


k=6

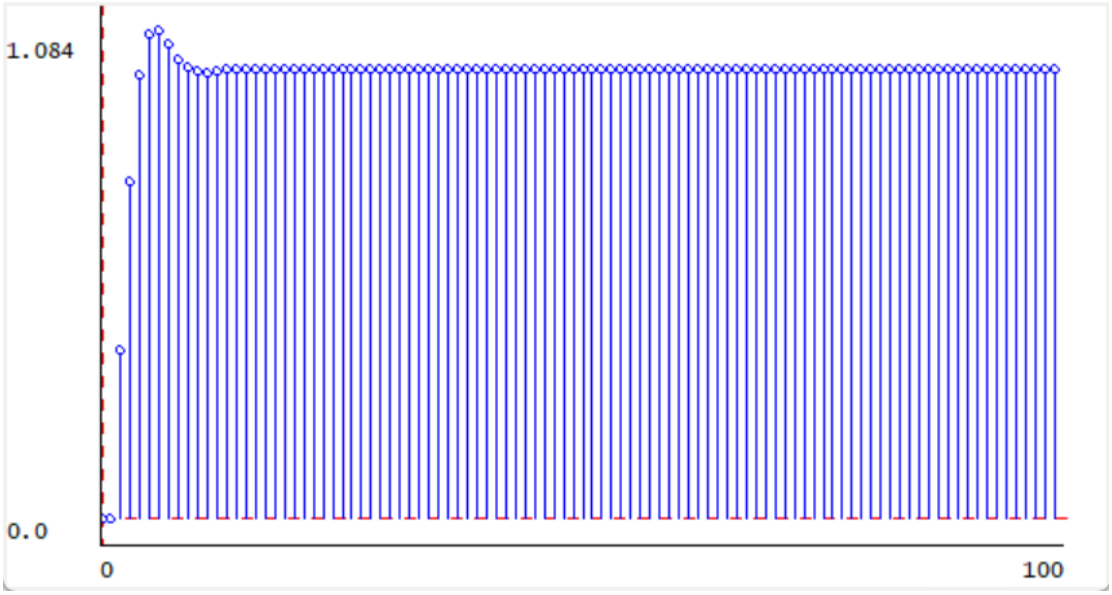


k=30

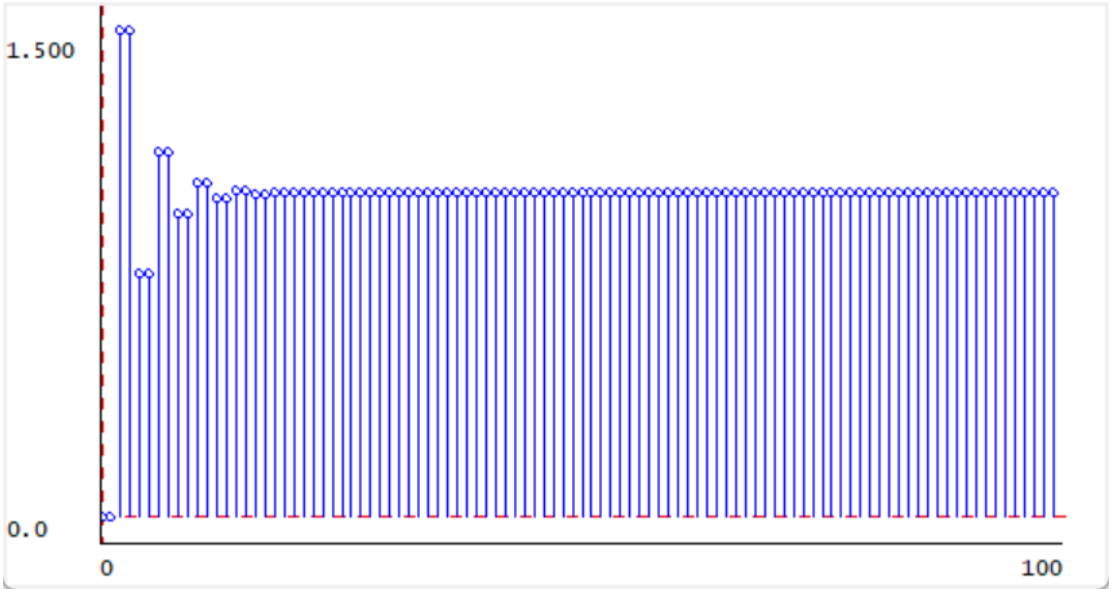
Take kc = 0.93775 and examine the impact of different T values on the system response. Starting from T = 0.005s, gradually increasing the T value, it can be found that the system will reach stability faster, but the oscillation will also gradually become more intense. Finally, it evolves into oscillation divergence.



T=0.02



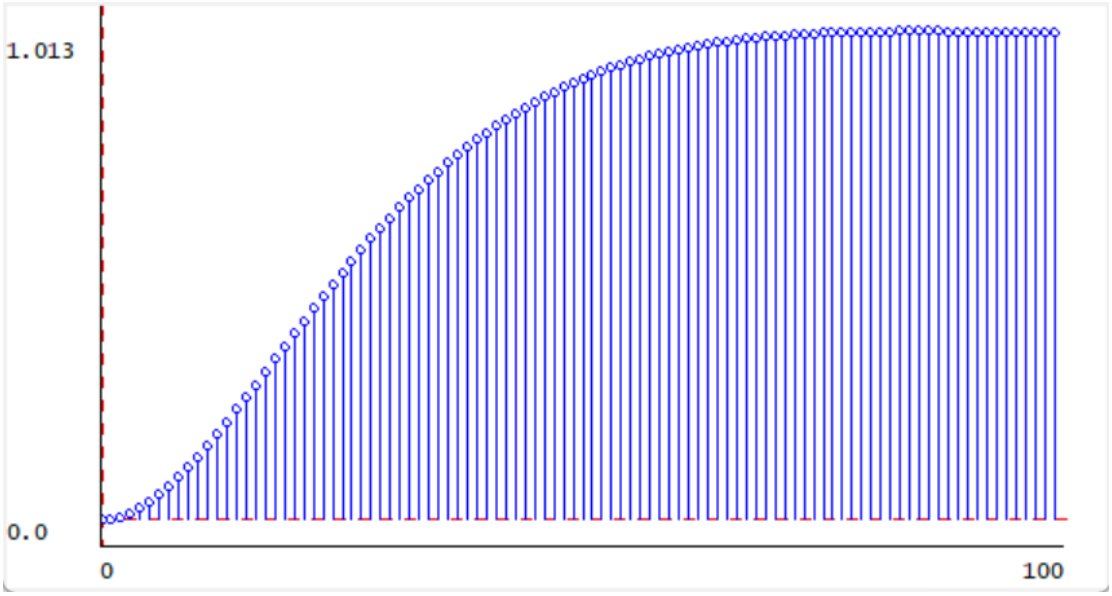
T=0.04



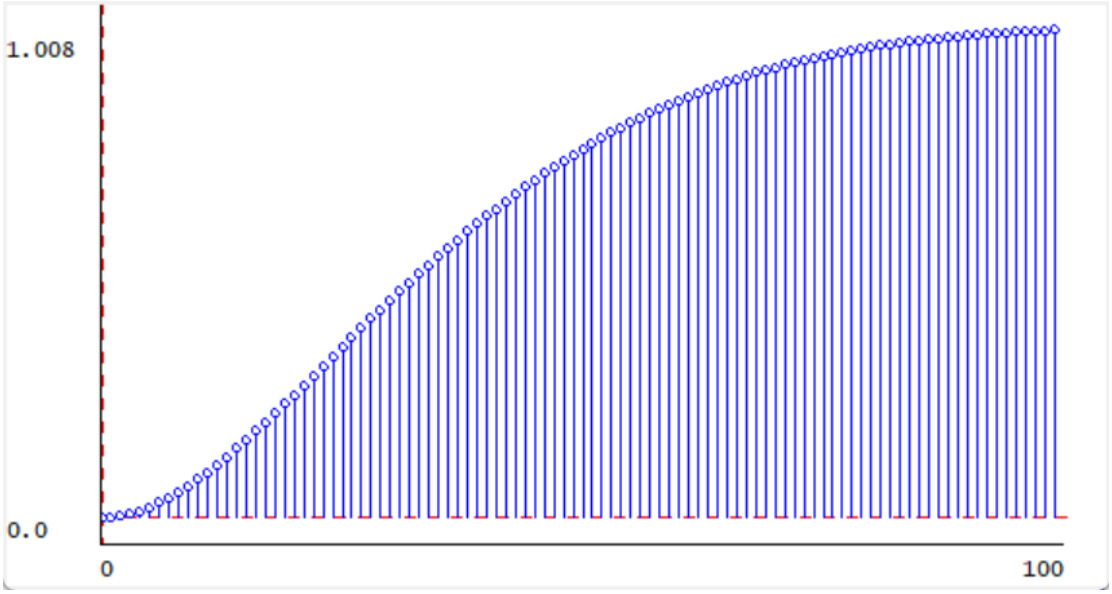
T=0.08

T=0.15

On the contrary, starting from T = 0.005s and gradually decreasing the T value, it can be found that the time for the system to reach a steady state becomes longer.



T=0.004



T=0.003

Optimal Solutions

Use optimize.optOverLine to find the minimum of one of them. The function f has a minimum at x = 0.5 of value −0.25; the function h has a minimum at x = 1.66 with value −0.88.

```
Python
import lib601.optimize as optimize
def f1(x):
    return x*x - x

def f2(x):
    return x**5-7*x**3+6*x**2+2

def Finder(f, min, max, numsteps):
    print (optimize.optOverLine(f , min, max, numsteps))

Finder(f1 , -5, 5, 1000)
Finder(f2 , 1, 2, 100)

# >>> (-0.25, 0.499999999999938)
# >>> (-0.8815419423999984, 1.6600000000000006)
```

Summary

- The experiment aimed to design a control system that adjusts the position of a robot head to align with a light source. Key components included a light sensor, motor, and proportional controller.

- The light sensor detected the angular error between the light source and head position, while the proportional controller converted this error into a control voltage to direct the motor.

- Python code was written to model each component, with specific attention to the controller gain (kc) and time interval (T) for evaluating system stability and response.

- Results showed that higher T values increased the system's response speed but led to oscillations, while lower T values ensured a stable convergence but slowed the system.

- The value of kc also impacted the response: appropriate kc values resulted in stable convergence, while others caused oscillatory behavior or divergence.

- Overall, the experiment demonstrated how tuning kc and T can balance stability and responsiveness, providing insights into optimal control parameter selection for robotic applications.